@AndrzejWasowski
Andrzej Wąsowski

@zhoulaifu
Zhoulai Fu

Advanced Programming

Parallelism (retrospective)

IT UNIVERSITY OF COPENHAGEN

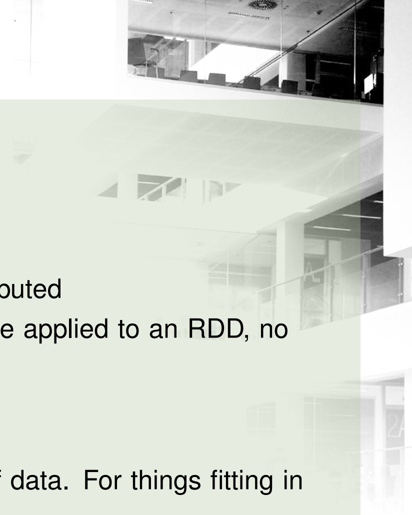
SOFTWARE
QUALITY
RESEARCH

Parallel Data Structures

- Consider a regular list: `val list=(1 to 100).toList`
- And a regular list computation: `list map f`
- What if `f` is very slow but referentially transparent?
- We can parallelize the mapping! (recall `parMap` from class)
- In Scala standard library we can: `list.par.map (f)`
- `list.par : collection.parallel.immutable.ParSeq[Int] =ParVector(1,...`
- Scala has parallelized versions for `ParArray`, `ParVector`, `mutable.ParHashMap`, `mutable.ParHashSet`, `immutable.ParHashMap`, `immutable.ParHashSet`, `ParRange`, `ParTrieMap`
- **Similarities** with `Par`: enable parallelism at the level of processing data structures without low level concurrency primitives (**parallel programming for the masses!**)
- **Differences** from `Par`: Scala's parallel collections are **eager**. We separate construction of the computation from execution. This gives more flexibility.
- Similar facilities exist in LINQ (C#) and in F#

`http://docs.scala-lang.org/overviews/parallel-collections/overview.html`

Parallel Collections in Spark



- Spark has seemingly similar facilities:

```
val data =Array(1, 2, 3, 4, 5)
val distData =sc.parallelize(data)
```

- Constructs an `RDD` from a collection.
- RDD resembles a parallel collection, but it can **also** be distributed
- **RDD constructions are lazy**. As long as transformations are applied to an RDD, no computation is executed.
- Allows Spark schedulers to control the computation better
- This is more like `Par` than Scala’s native parallel collections
- NB. `parallelize` will only be faster than par, if we have a lot of data. For things fitting in memory of a single computer it is likely slower.

http://spark.apache.org/docs/latest/programming-guide.html#parallelized-collections